KAoS: A Knowledgeable Agent-oriented System

Nicholas J. George
Robert J. Jasper
Monica Rosman LaFever
Katrina M. Morrison
Daniel B. Rosenthal
Stephen R. Tockey
John D. Woolley
Seattle University

Jeffrey M. Bradshaw
European Institute of Cognitive Sciences and Engineering (EURISCO) and The Fred Hutchinson Cancer Research Center

Guy A. Boy
European Institute of Cognitive Sciences and Engineering (EURISCO)

Peter D. Holm
The Boeing Company

Abstract

In this paper, we discuss an effort to develop and evaluate a generic agent framework. A major goal of this work was to separate generic agent characteristics from domain-specific characteristics. To test out our ideas, we implemented a demonstration prototype called KAoS (Knowledgeable Agent-oriented System) in Smalltalk-80 on the Macintosh, extending the Smalltalk primitives in C.

We developed the original system to support local and distributed interapplication facilities using Apple Events, Program-to-Program Communication (PPC), Dynamic Data Exchange (DDE), and TCP/IP protocols. We are currently extending the architecture with: a more robust object interoperability foundation (Common Object Request Broker Architecture or CORBA), agent authoring and run-time interfaces, and learning and adaptive mechanisms.

1 Introduction

Based on experience and recent studies, we expect that automation of dynamic, real-time environments such as air-traffic control will dramatically increase in the coming decades [Billings, 1991], [Boy and Hollnagel, 1993]. The complexity, real-time constraints, and distributed nature of such tasks require that software not merely respond to requests for information but intelligently anticipate, adapt, and actively seek ways to support users. Not only must these systems assist in coordinating tasks among humans, they must also help manage cooperation among distributed programs.

[Brodie, 1989] has discussed the need for *intelligent interoperability* in such environments. He defines the term to mean intelligent cooperation among systems to optimally achieve specified goals. A high level of interoperability requires knowledge of the capabilities of each system, so that task planning, resource allocation, execution, monitoring, and, possibly, intervention between the systems can take place. Ideally, an intelligent agent would function as a global resource planner.

While a single agent would be workable for small networks of systems, such a scheme would quickly become impractical as the number of cooperating systems grows. The activity of the global resource planner would become a bottleneck for the (otherwise distributed) system. A further step toward intelligent interoperability would be to embed one or more agents within each cooperating system. Applications would request resources through these agents, thus providing a level of encapsulation at

the planning level, analogous to the encapsulation provided in many existing applications at the level of data exchange protocols.

An equally important contribution of research on agents may not have to do with any direct technical advantage they afford, but rather with the fact that taking this perspective helps us think about our software differently. Just as an object-oriented representation more easily and clearly expresses some algorithms than a procedural representation [Kaehler and Patterson, 1986], so it sometimes may be easier to use agents rather than objects. The agent metaphor encourages the user to take an *intentional stance* [Dennett, 1987], [Sharp, 1993] in predicting the behavior of the system. This hopefully allows the user to better cope with the increasing complexity and dynamic nature of software.

In this paper, we discuss an effort to develop and evaluate a generic agent framework. The framework promotes more efficient and coherent communication among people and their programs. After a general overview of the project and architecture (sections 2.1–2.3), we discuss the characteristics that are common to all KAoS agents (section 2.4). In response to new application requirements, we are extending the architecture with a more robust object interoperability foundation, agent authoring and run-time interfaces, and learning and adaptive mechanisms (section 3).

2 KAoS Overview and Architecture

KAoS (Knowledgeable Agent-oriented System) is part of a multiyear project to develop a distributed environment for software agents. Specifically, KAoS provides an infrastructure for programming agents that includes: several network communications tools, distributed messaging, an agent communications protocol, and generic agent class hierarchy, shell, and controls.

We built KAoS using Smalltalk-80 v. 4.1 and C on the Macintosh. We augmented the underlying Smalltalk interpreter to perform distributed agent-to-application communication. Work on a portable version that will run on Unix and Microsoft Windows platforms as well is currently underway (see section 3.1).

We initially used KAoS to build a demonstration of agent-oriented programming and to simulate various agent activities. We are now enhancing it for use in two applications:

- real-time collaboration, information management, and decision support for the treatment of bone-marrow cancer patients [Bradshaw et al., 1993a].
- adaptive mobile performance support of aircraft maintenance personnel [Bradshaw et al., 1993c].

These applications have provided both a rich source of requirements for the KAoS architecture, as well as a number of interesting new problems to solve.

Figure 1 identifies important elements of the original KAoS architecture. We discuss each of them in turn.

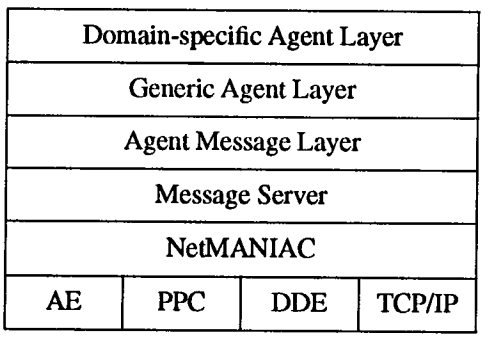

Figure 1. KAoS Architecture Elements.

2.1 NetMANIAC

The Network Manager for Interapplication Communication (NetMANIAC) provides the foundation for the KAoS architecture. It provides local and distributed interapplication communication facilities using Apple Events (AE), Program-to-Program Communication (PPC), Dynamic Data Exchange (DDE), and TCP/IP protocols.

2.2 Message Server

The Message Server provides a transparent interobject message forwarding facility. It allows any object to exchange messages with any other named object without knowing the physical location of that object.

2.3 Agent Message Services

The agent message layer provides classes and methods for dealing specifically with agent messages. This includes methods for distinguishing between classes of messages that agents can send. Because the Message Server provides mapping of names to locations, agents need not know where the agents they are communicating with reside.

2.4 Generic Agent Functionality

The Generic Agent facility provides a framework for constructing agents. KAoS provides a 'generic agent' class from which you can specialize agents using inheritance. The generic agent class includes default behaviors, and includes classes and methods for representing and updating an agent's mental state. You create agents of arbitrary complexity and capabilities using this base class; this provides inheritance of general characteristics that allow agents to be productive and communicate effectively with other agents. More importantly, programmers can create hierarchies of agents, each layer specifying more specialized behavior and innate knowledge.

A major goal of KAoS is to separate generic agent characteristics from domain-specific characteristics. Our work is a logical extension of the agent-oriented programming paradigm [Shoham, 1990]. From our work, we concluded which characteristics you can abstract and incorporate at the generic agent level. Separating out generic characteristics provides a powerful framework from which to create agents. We present the following views of these common characteristics:

- External view—how agents communicate
- Structure—what agents are composed of
- Dynamics—how agents behave over time
- Properties—constraints on agent behavior

2.4.1 External View of Agents

We have modeled the agent-to-agent protocol on a small number of generic speech acts that you can extend and specialize for particular domains. The fundamental unit of agent-to-agent communication is a transaction. Transactions are sequential sets of messages that elicit cooperation or transfer knowledge between agents. Transactions always terminate with either an accept or reject message.

The KAoS architecture includes mechanisms for controlling communications between agents, storing the context of transactions, and avoiding repetitive and unproductive communication between agents. We based the design of the transaction protocol on *petitions* and *responses*. Petitions are entreaties made from one agent to another. Generic petitions consist of:

- Inform—sender asks receiver to update the receiver's knowledge
- Request—sender asks receiver to make a commitment to the sender.
- Cancel—sender asks receiver to remove a previous commitment.
- Refrain—sender asks receiver to not make a particular commitment.

The set of petitions is identical to those described in [Shoham, 1990] and [Thomas et al., 1991]; we use the term *petition* to refer to these actions in a more general way.

Responses are the answers returned by the receiver of a petition:

- Accept—sender accepts the petition, which ends the transaction.
- Reject—sender refuses to accept the petition, which ends the transaction.
- Counter—sender refuses to accept the petition, but responds with an alternative. It is up to the petitioner to consider the alternative offer.

[Shoham, 1990] and [Thomas et al., 1991] do not use counters, but [Bowers and Churcher, 1988] include them in the protocol description. It was necessary to add counters to facilitate negotiation between agents. [George et al., 1993] gives further details on the protocol and differences from previous work.

The protocol and architecture allow agents to participate in multiple transactions at the same time with different agents. The architecture does not allow embedded transactions; an agent cannot petition another agent while involved in a transaction as petitioner with that same agent. This restriction did not appear to be a problem. It was both convenient and useful to support parallel transactions; two agents can participate in two transactions with each as petitioner (see Figure 2).

The architecture also supports maintaining an entire transcript of the transaction; this is useful during the transaction to keep track of who said what. The transcript along with a numeric counter for 'counter' messages helps avoid unproductive or repetitive messages (and provides some guarantee that the transaction will eventually terminate).

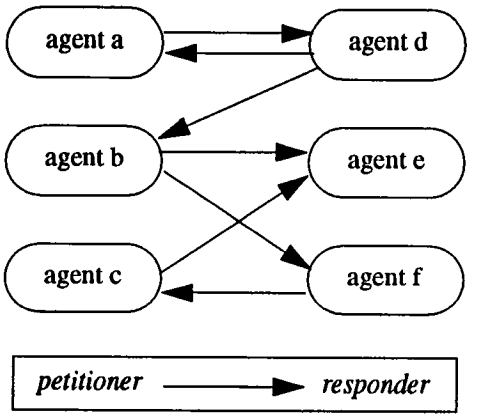

Figure 2. Multiple and Parallel Transactions.

2.4.2 Agent Structure

The KAoS architecture and generic agent class provide a consistent structure for agents; this includes mechanisms for storing, updating, and inheriting: knowledge, commitments, choices, and capabilities. Figure 3 shows these structures.

Knowledge is a collection of facts and beliefs. Facts are simply beliefs in which the agent has complete confidence. Commitments are obligations the agent has made. Choices are simply obligations the agent has made to itself. Capabilities are an exported list of those services or functions that an agent can provide.

Our original design included similar mechanisms for agent desires. We concluded that desires tend to be so domain-specific that there is little value in creating generic structures and mechanisms for them. However, we found it easy to program agents with domain-specific desires using the structure already provided by knowledge, commitments, capabilities, and choices. The current design provides mechanisms for inheriting desires within the agent hierarchy.

2.4.3 Agent Dynamics

Because agents are autonomous entities, each agent goes through birth, life, and typically death. At birth, agents are instantiated with specific innate knowledge and capabilities. During their lives, agents go through a continuous cycle of reading messages (communicating), fulfilling their desires, and discharging any commitments due at the current time. Agents may acquire additional knowledge and capabilities as they interact with other agents and their environment.

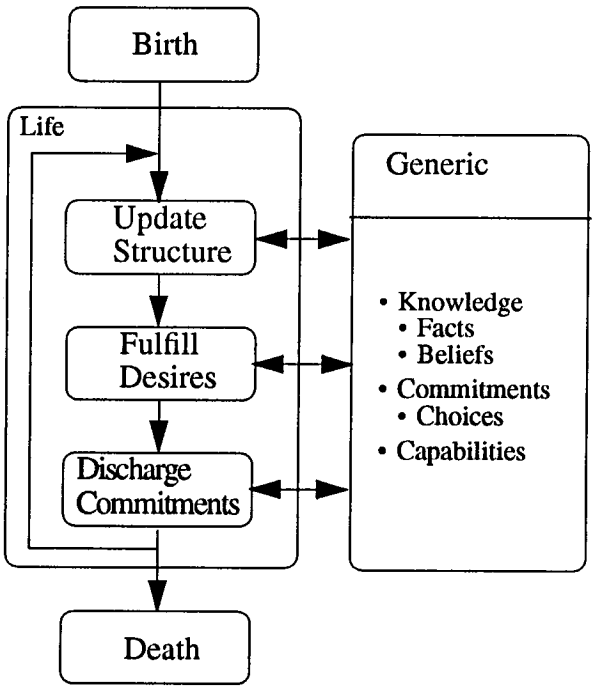

Figure 3. Structure and Dynamics of Agents.

Some applications may require agent death to free resources or simply deal with agents that are no longer useful. Agent death poses special problems. Depending on the application, it may be necessary to include domain-specific procedures for dealing with agent death. These may include notification of other agents, transfer of any pending commitments, or transfer of knowledge.

2.4.4 Agent Properties

There are several general properties the KAoS architecture provides [Shoham, 1990]:

- Internal consistency—agents should have no conflicts in their mental state. Updates to an agent's mental state can only occur through mechanisms the architecture provides. These mechanisms include consistency checks to assure each agent's mental state is consistent with itself.
- Rationality—agents should attempt to fulfill their desires. The overall control structure shown in Figure 3 provides this property.
- Good faith—agents should only commit to things that they are capable of and intend to carry out. Because each agent has a published list of capabilities, the agent will not accept any requests that it can not carry out.
- Introspection—agents should be aware of their own mental state. KAoS provides each agent

with full access to its own mental state. It can only obtain information about another agent's mental state through a request.

- Persistence of mental state—agents should not change their mental state randomly. Update to an agent's mental state can only occur through mechanisms the architecture provides (e.g., inform messages, inference).

2.5 Domain-specific Agents

The Domain-specific Agent layer allows users of KAoS to create specialized agents. As part of our initial application, we implemented several domain-specific agents. One particularly useful class of agents developed was the service broker.

Service broker agents function to mediate between client agents and external resources and services. We provided two additional layers of abstraction between the service broker and the actual application. The service broker handed requests to a resource manager who handled particular kinds of services (e.g., database applications). Resource managers in turn forwarded requests to an application manager who controlled a specific application (e.g., Oracle).

3 Current Directions

While the original design of KAoS was sufficient for testing a number of basic concepts, we did not intend it to be a robust and complete implementation of an agent-oriented programming and run-time environment. The requirements of our medical and aerospace performance support applications have motivated extensions in three main areas:

1. Incorporation of the OMG Common Object Request Broker Architecture (CORBA).
2. Agent authoring and run-time interfaces.
3. Learning and adaptive mechanisms.

3.1 Incorporation of CORBA

Since the time of the first KAoS implementation, Hewlett-Packard has produced a Smalltalk-80 implementation of the OMG Common Object Request Broker Architecture (CORBA) called "Distributed Smalltalk." This environment allows transparent access to objects and methods not only among different Smalltalk environments, but also among the growing number of CORBA-compliant applications. Distributed Smalltalk uses a Berkeley sockets mechanism that ParcPlace Systems implements for Unix, Macintosh, and Microsoft Windows environments.

As part of a follow-on effort called agONy!, we are currently reimplementing NetMANIAC and the Message Server to take advantage of the capabilities of Distributed Smalltalk. Along with the Berkeley Sockets mechanism, we will support Apple Events, DDE, PPC, and eventually OpenDoc and OLE 2.0, so that data and services from Macintosh and Windows applications are available to KAoS.

The adoption of an open standard for object interoperability as the foundation for KAoS will make it easier for us to port subsets of the KAoS architecture to other CORBA-compliant programming environments and applications.

3.2 Authoring and Run-time Interfaces

In the original version of KAoS, you could develop and monitor agents only through Smalltalk browsers and rudimentary run-time interfaces. In our current work, we are applying principles from knowledge acquisition, end-user authoring, and programming-by-demonstration research to develop and evaluate more adequate tools [Bradshaw et al., 1993b], [Cypher, 1993], [Erickson, 1991], [Laurel, 1990], [Nardi, 1993], [Oren et al., 1990], [Spohrer et al., 1991].

In this work, we are interested in knowing how the use of agents affects the way we perceive and use the system. Users approach interfaces with a host of expectations and assumptions, based on their previous experience with computers and the specific presentation of the system. Metaphors consistent with the everyday experience of users can speed their learning of computing conventions. For example, the desktop metaphor takes advantage of users' previous knowledge that office artifacts are visible, are passive, have locations, and may contain things. Ontological knowledge of a different sort comes into play when you employ the agent metaphor [Erickson, 1991]. For example, users assume that agents can hide, can initiate things, can go places, and can make choices and commitments. They may also assume less tangible qualities like that they can know things, can have goals and intentions, they are internally consistent, rational, act in good faith, can introspect, can cooperate to achieve common goals, and that their mental state persists.

As we develop authoring and run-time tools for agents, it is important to evaluate their effectiveness from the end-user's point of view.

3.3 Learning and Adaptive Mechanisms

In complex dynamic environments, it is very difficult to anticipate situations or contexts that agents will encounter. Thus, these environments must perform knowledge acquisition of situation patterns and appropriate behavior online and incrementally as agents perform tasks in the real world. We are providing a framework for the acquisition of situational knowledge by reimplementing and refining the Situation Recognition and Analytical Reasoning (SRAR) model and its associated "block" knowledge representation [Boy, 1991], [Boy and Mathé, 1993]. The SRAR model was originally developed in 1986 as part of a project to aid astronauts in diagnosing faults in the orbital refueling system of NASA's space shuttle. It has subsequently been applied at NASA-Ames to develop a suite of Computer-Integrated Documentation (CID) applications [Boy, 1992]. Working in collaboration with researchers at NASA-Ames, we will integrate selected CID contextual learning mechanisms into the KAoS architecture to assess their value in performance support applications.

The SRAR model provides a formal framework for integrating situational (problem-statement situational patterns) and analytical (problem-solving resources) knowledge (Figure 4). In the beginning, intelligent systems are "inexperienced" and must rely on broad analytic knowledge. Learning mechanisms in such systems rely on the reinforcement of successful actions, the discovery of failure conditions, and the generation of recovery actions to improve performance. Elements of the analytical knowledge are transferred into situation patterns. Over time, situational patterns multiply and become more complex, while analytical knowledge becomes more structured.

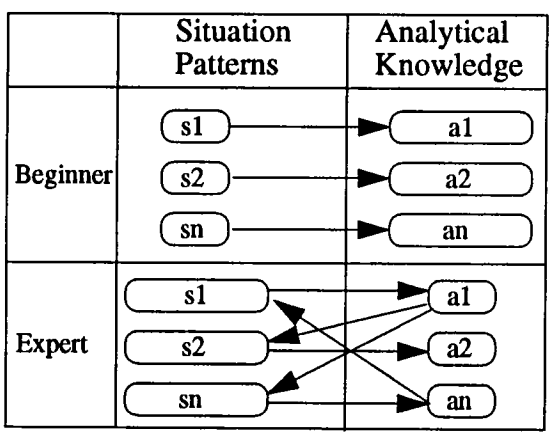

Figure 4. The SRAR Model.

Individual and collective agent knowledge is represented as networked "blocks." The block representation permits agents to interactively acquire knowledge about relationships between pieces of relevant information and the problem context. A symbolic learning method transfers elements of the analytical knowledge into situation patterns, and subsequently assists with knowledge base restructuring. Most of the learning mechanisms work in the background, requiring minimal overhead effort by users. The result over time is a set of agents that have learned by experience how to adapt to particular users and situation patterns.

4 Conclusions

There is every reason to expect that future applications will need to intelligently anticipate, adapt, and actively seek ways to support users. A mix of object-oriented and agent-oriented programming paradigms appears to be a fruitful area of future application and research. Based on our experience, an architecture for programming agents by inheriting from more general and generic agents appears to provide many benefits. Further work being carried out in the context of current applications will provide feedback on the extensibility and generality of the architecture.

References

[Billings, 1991] C. E. Billings. *Human-centered Aircraft Automation: A Concept and Guidelines.* NASA Technical Memorandum 103885, August 1991.

[Bowers and Churcher, 1988] J. Bowers and J. Churcher. Local and global structuring of computer-mediated representation: Developing linguistic perspectives on CSCW in Cosmos. *Proceedings of the Conference on Computer-supported Cooperative Work,* Portland, OR.

[Boy, 1991] G. A. Boy. *Intelligent Assistant Systems.* Academic Press, London, England.

[Boy, 1992] G. A. Boy. Computer integrated documentation. In E. Barrett (ed.), *Sociomedia: Multimedia, Hypermedia, and the Social Construction of Knowledge*, pages 507–531, MIT Press, Cambridge, MA.

[Boy and Hollnagel, 1993] G. A. Boy and E. Hollnagel. *Cockpit Evolution and Human-centered Automation.* European Institute of Cognitive Sciences and Engineering (EURISCO), September 1993.

[Boy and Mathé, 1993] G. A. Boy and N. Mathé. Operator assistant systems: An experimental approach using a telerobotics application. In K. M. Ford and J. M. Bradshaw (eds.), *Knowledge Acquisition as Modeling,* pages 271–286. Wiley, New York, NY.

[Bradshaw et al., 1993a] J. M. Bradshaw, C. R. Chapman, K. M. Sullivan, J. H. Boose, R. G. Almond, D. Madigan, D. Zarley, J. Gavrin, J. Nims, and N. Bush. KS-3000: Applying DDUCKS to bone-marrow transplant patient support. *Proceedings of the Seventh European Knowledge Acquisition for Knowledge-based Systems Workshop* (EKAW-93), Toulouse and Caylus, France, July 1993.

[Bradshaw et al., 1993b] J. M. Bradshaw, K. M. Ford, J. R. Adams-Webber, and J. H. Boose. Beyond the repertory grid: New approaches to constructivist knowledge acquisition tool development. In K. M. Ford and J. M. Bradshaw (eds.), *Knowledge Acquisition as Modeling,* pages 287-333, Wiley, New York, NY.

[Bradshaw et al., 1993c] J. M. Bradshaw, T. Richards, P. Fairweather, C. Buchanan, R. Guay, D. Madigan, and G. A. Boy. New directions for computer-based training and performance support in aerospace. *Proceedings of the Fourth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace*, Toulouse, France, 28–30 September 1993.

[Brodie, 1989] M. L. Brodie. Future intelligent information systems: AI and database technologies working together. In J. Mylopoulos and M. L. Brodie (eds.), *Readings in Artificial Intelligence and Databases*, pages 623–642. Morgan Kaufmann, San Mateo, CA.

[Cypher, 1993] A. Cypher (ed.). *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA.

[Dennett, 1987] D. C. Dennett. *The Intentional Stance.* MIT Press, Cambridge, MA.

[Erickson, 1991] T. Erickson. *Design Issues for Enhanced Interfaces: Models and Metaphors.* Apple Computer Human Interface Group, Cupertino, CA, 26 November 1991 (draft).

[George et al., 1993] N. J. George, R. J. Jasper, M. R. LaFever, K. M. Morrison, D. B. Rosenthal, and S. R. Tockey. KAoS agent-to-agent protocol description. In *KAoS: Knowledge Agent-oriented System*, Volume 4, Seattle University, Seattle, WA.

[Kaehler and Patterson, 1986] T. Kaehler and D. Patterson. A small taste of Smalltalk. *BYTE*, pages 145–159, August 1986.

[Laurel, 1990] B. Laurel. Interface agents: Metaphors with character. In B. Laurel (ed.), *The Art of Human-Computer Interface Design*, pages 355–366. Addison-Wesley, Reading, MA.

[Nardi, 1993] B. A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing.* MIT Press, Cambridge, MA.

[Oren et al., 1990] T. Oren, G. Salomon, K. Kreitman, and A. Don. Guides: Characterizing the interface. In B. Laurel (ed.), *The Art of Human-Computer Interface Design*, pages 367–382. Addison-Wesley, Reading, MA.

[Sharp, 1993] M. Sharp. *Reactive Agents.* Technical Report. Apple Computer, Cupertino, CA.

[Shoham, 1990] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, (60)1:51–92, March 1993.

[Spohrer et al., 1991] J. C. Spohrer, D. Vronay, and R. Kleiman. Authoring intelligent multimedia applications: Finding familiar representations for expressing knowledge. *Proceedings of the 1991 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1751–1756, Charlottesville, VA, 13–16 October, 1991.

[Thomas et al., 1991] B. Thomas, Y. Shoham, A. Schwartz, and S. Kraus. Preliminary thoughts on an agent description language. *International Journal of Intelligent Systems*, 6(5):497–508, August 1991.